New Jersey Institute of Technology

A. V. Gerbessiotis

CIS 750

Jan 30, 2006

Spring 2006

(150 points) Due: Feb 27, 2006

Programming Assignment 1

1 Introduction

Before you do the assignment it is imperative that you read Handout 2, familiarize yourself with the account setup and the test platform, and decide what tools you are going to use.

We first give an outline of the assignment.

You are asked to implement the first part of the indexing that a search engine performs after a Web-crawler has collected Web-pages from the Web. This task is accomplished in three parts.

Searchable Documents You traverse the local directory structure of the locally stored Web-pages and identify the searchable files among them. Some assistance is provided in the protected area of the course Web-page on how to traverse a directory structure in a UNIX file-system. You can use this information as a base for this part of the assignment. For the sake of this assignment searchable files will be uncompressed ASCII text files. After you have identified the searchable text files, you are then about to start the process of parsing these files and tokenizing their contents.

Tokenization The second part involves the first phase of parsing, that of tokenization. For every searchable document, you identify and extract keywords/tokens of interest along with other useful information (such as word position in the text, font-size, and position in an html document). In this part you also convert keywords to numbers (wordids) and also URL locators to numbers (docids).

Linguistic Analysis The third part involves the second phase of parsing, the linguistic analysis of the keywords/tokens that would be turned into index terms. For the sake of this assignment we will concentrate only on elementary `stopword` removal. The output of this phase will be then fed into the indexer.

This is the minimum implementation required to gain you the `150` points of this assignment. You can enrich this implementation but adding additional features. Or you can think ahead of adding even more feature and components (eg. stemming) in the context of the independent work component of the course.

2 Deliverables

The relevant files will become available in `homework/pa1` as described in Handout 2. A single executable file will be the result of your compilable source code into the form of a file named `pa1`.

3 Part 1 : Searchable Document

The executable file `pa1` will read the command line and behave as follows.

```
% ./pa1 slist file-name
% ./pa1 slist directory-name
```

The first argument in the command line (after the name of the executable) denotes the action. The second argument is a file/directory name.

For action `slist` (searchable document list) you need to decide whether `file-name`, if file-name is the name of a file, or the files under the `directory-name`, if directory-name is a directory, are searchable text files or not and list them.

A searchable text file `STF` is a file with one of the following suffixes:

`.html , .htm , .txt , .cc , .cpp , .c , .h , .java .`

An `STF` file can be `HTML` or `TEXT`. An `HTML` file is a searchable text file with suffixes `.html` or `.htm`. A `TEXT` file is a searchable text file with any of the remaining suffixes. A file that is either a directory name or a non-searchable file is `NSTF` (non-STF).

The command argument `slist` prints for every file identified as an `STF` whether it is `HTML` or `TEXT` along with the full path name relative to the directory name listed as the second argument during program invocation. A (partial) output may look like as follows in the case where there are two HTML files in subdirectory `cis435` of directory `courses` which is the only directory of `alexg`.

```
% ./pa1 slist  alexg
  alexg/courses/cis435/index.html  HTML
  alexg/courses/cis435/handouts.html HTML
  END-OF-LISTING
```

Note that if you call the program with a directory name prefix that includes `/home` and a user-account such as `u20` the whole prefix is trimmed away and replaced by the default URL as explained earlier.

```
% ./pa1 slist  /home/u20/alexg
```

Therefore this latter invocation will also generate the same output as before.

4 Part 2 : Tokenization

If action is `token` then every file is identified as before as an `STF` or `NSTF` (but not printed in the output) with the additional requirement that for every `STF` file a parsing function is called that tokenizes each file into its individual tokens.

```
% ./pa1 token file-name
% ./pa1 token directory-name

% ./pa1 debug-token file-name
% ./pa1 debug-token directory-name
```

The tokenization phase is probably the most difficult part of this assignment and the most time consuming. You need to decide how to parse a document and what constitutes a token. The choice of C/C++/Java is up to you. However you might make the job at hand easier if you explore using lex or yacc for this part. An hour or two reading a manual page or the extensive documentation for the GNU equivalents names flex and bison might save you more time later.

Tokenizing a `TEXT` file is easier. Interesting tokens that will become index terms are going to be words (eg. alphanumeric strings starting with a character) or non-trivial numbers. Single digit numbers can be discarded dates such as 1950 might become useful searchable terms. Collectively we will call all these interesting tokens `words` even if some of them are numbers. **The tokenizer generates words that are lower case no matter what the original case was.**

`words` will be represented by `wordid`s, i.e. a preferably 32-bit unsigned number.

`Documents` will be represented by `docid`s, i.e. a preferably 32-bit unsigned number.

You need to have ways to determine the word represented by say `wordid` 25 or the full URL name of the document with `docid` 100.

Note that for the sake of this and remaining assignments a file in say directory
`alexg/courses/cis435/index.html` has a full URL which is
`http://www.cs.njit.edu/~alexg/courses/cis435/index.html`.

Therefore if word `algorithms` is identified in a `TEXT` document a quadruplet of information is output by the `tokenizer`:
`(docid,wordid,wpos,attr)`

The first element of the quadruplet is a `docid` and not the full-URL of the document in which the named word appears. The second one is the unique `wordid` for the specified word (eg. `algorithms`). All files that have been searched should have a common lexicon and a single entry for each unique word. The third attribute is positional information about the identified word in the document. It identifies the word position of the word in the document, i.e. whether `algorithms` is the 10-th or 20-th word in the named document. The last element is an `attribute` value. For `TEXT` files all words have atttribute 0. For `HTML` files certain words may have higher attribute values. Words surrounded by a `<TITLE>` tag will have attribute value 1. Words surrounded by an `<A>` anchor tag will have attribute value 2. Words surrounded by a `<H1> to <H3>` tag have value 3, and `<H4> to <H6>` tags have value 4. Other values are possible; your implementation is open-ended.

So a command such as the one depicted below will generate in the output a stream of parenthesized numbers

```
% ./pa1 token alexg/courses/cis435/index.html
(10,20,30,4)
```

Note that we do not specify how big the attribute or positional values are going to be. It's up to you to decide so.

If however the command is `debug-token` more useful information can be printed such as

```
% ./pa1 debug-token alexg/courses/cis435/index.html

( http://www.cs.njit.edu/~alexg/courses/cis435/index.html,algorithms,30,<H4>)
```

5 Part 3 : Linguistic Analysis

The linguistic analysis module will take as input the output of Part 2 and eliminate those keywords that are identified as stopwords. The list of stopwords for the sake of this assignment is given below.

```
   I a about an are as at be by com en for from how in is it  of on or that the
this to was what when where who will with www
```

```
% ./pa1 stopwords file-name
% ./pa1 stopwords directory-name
```

The effect of command `stopwords` is incremental. It is equivalent to `token` with the addition of stopword elimination.

6 Side Effects

Each one of the three commands `stopwords`, `token` and `debug-token` will have the following common side-effect. The generation of a file `lexicon` in the directory in which `pa1` was issued and the generation of a file `doclist` also in the same directory. The first file `lexicon` lists all words indexed by docid and the latter `doclist` lists all URL/documents indexed also by doc id. The two files are ASCII printable/viewable files. For example the output might look like as follows. No indentation or pretty printing is required. However one item per line should be printed.

```
% cat ./lexicon
 0  alex
 ...
 20 algorithms
 ....
```

■